T1

The program counter contains the address of an `LDR` instruction. In order for the LC-3 to process that instruction, how many memory accesses must be made? Repeat this task for `STI` and `TRAP`.

> answer:
>
> `LDR`：取指令 1 次，读取数据 1 次，共 2 次。
>
> `STI`：取指令、读指针地址的到最终地址、写入内存各 1 次，共 3 次。
>
> `TRAP`：取指令 1 次，从向量表读出返回地址 1 次，共 2 次。

T2

We would like to have an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called NOP, for NO OPERATION. the instruction still goes through the six phases of the instruction cycle but the execution phase is to do nothing! Please provide a machine instruction using `ADD`, `AND` and `BR` respectively that could be used for `NOP` and have the program still work correctly. You can use the register 1.

> answer:
>
> `ADD: 0001 001 001 100000`
>
> `AND: 0101 001 001 111111`
>
> `BR: 0000 000 000000000` // 合理即可

T3

The PC contains x3010. The following memory locations contain values as shown:

```
x3050:    x70A4
x70A2:    x70A3
x70A3:    xFFFF
x70A4:    x123B
```

The following three LC-3 instructions are then executed, causing a value to be loaded into R6. What is that value?

```
x3010    1110 0110 0011 1111
x3011    0110 1000 1100 0000
x3012    0110 1101 0000 0000
```

We could replace the three-instruction sequence with a single instruction. What is it?

> answer:
>
> ```
> x3010    LEA R3 x003F    // R3 <- x3011+x003F=x3050
> x3011    LDR R4 R3 #0    // R4 <- Mem[x3050]=x70A4
> x3012    LDR R6 R4 #0    // R6 <- Mem[x70A4]=x123B
> ```
>
> 最终 `R6` 的值是 x123B.
>
> 我们可以直接使用间接访存指令 `LDI` 一步达成：
>
> `x3010    1010 1100 0011 1111    // LDI R6 x003F`
>
> 指令操作为 `R6 <- Mem[Mem[x3011+x003F]]`

T4

What is the difference between the following LC-3 instructions A and B? How are they similar? How are they different? A: `0000 111 101010101` B: `0100 1 11101010101`

> answer:
>
> 第一条指令为 `BR` 条件跳转指令，由于 `NZP` 条件码均为 1，一定发生跳转。第二条指令为 `JSR` 无条件跳转指令，两条指令均是 PC 相对寻址，偏移量均为 `xFF55=-x00AB`，均跳转到 `PC-x00AB`.
>
> 但 `BR` 为普通分支，不保存返回地址，而 `JSR` 用于子程序调用，会将返回地址保存至 `R7`.

T5

Please explain the addressing range of PC-relative mode, regarding the current instruction address as the origin.

> answer: $[1-2^8, 2^8]$

T6

State the contents of R0, R1, R2 and R3 after the program starting at location x3001 halts.

Adress	Data
x3001	1110 001 111111101
x3002	0001 011 001 100011
x3003	0101 010 001 000011
x3004	0011 010 000000001
x3005	1001 001 001 11111
x3006	1001 011 010 11111
x3007	1111 0000 0010 0101

> answer:

```
x3001    LEA R1 #-3    // R1 <- x2FFF
x3002    ADD R3 R1 #3  // R3 <- x3002
x3003    AND R2 R1 R3  // R2 <- x2002
x3004    ST  R2 #1     // M[x3006] <- R2
x3005    NOT R1 R1     // R1 <- ~R1
x3006    LD  R0 #2     // R0 <- x3009    由于 x3004 处指令，内容变化
x3007    HALT
```

最终，`R0: x3009`，`R1: xD000`，`R2: x2002`，`R3: x3002`

T7

When solving a complex problem, it is often necessary to break it down into several subtasks, implement them first, and then integrate them in a specific structure to achieve the overall goal. The most common fundamental structures are **sequence, condition, and iteration**.

1. Using the **sequential** construct, write a LC-3 machine language routine that calculates the 4th term of the Fibonacci sequence `F4`. Store the result in `R2`.

2. Using the **iterative** construct, write a LC-3 machine language routine that calculates the 10th term of the Fibonacci sequence `F10`. Store the result in `R2`.

> answer:
>
> 1.
> ```
> 0101 000 000 100000    // AND R0 R0 #0
> 0001 001 000 100001    // ADD R1, R0, #1
> 0001 000 000 000001    // ADD R0 R0 R1
> 0001 001 000 000001    // ADD R1 R0 R1
> 0001 010 000 000001    // ADD R2 R0 R1
> ```
>
> 2.
> ```
> 0101 000 000 100000    // AND R0, R0, #0
> 0001 011 000 100100    // ADD R3, R0, #4    // 计数
> 0001 001 000 100001    // ADD R1, R0, #1
> 0001 000 000 000001    // ADD R0 R0 R1
> 0001 001 000 000001    // ADD R1 R0 R1
> 0001 011 011 111111    // ADD R3 R3 #-1     // 递减计数器
> 0000 001 111111100     // BRp #-4           // 若大于0，循环继续
> 0001 011 000 000001    // ADD R2 R0 R1
> ```

T8

The LC-3 does not have an opcode for the logical function `OR`. That is, there is no instruction in the LC-3 ISA that performs the `OR` operation. However, we can write a sequence of instructions to implement the `OR` operation. The following four-instruction sequence performs the `OR` of the contents of register 1 and register 2 and puts the result in register 3. Fill in the two missing instructions so that the four-instruction sequence will do the job.

(1): `1001 100 001 111111`

(2):

(3): `0101 110 100 000 101`

(4):

answer:

$$A\ OR\ B = NOT(NOT A\ AND\ NOTB)$$

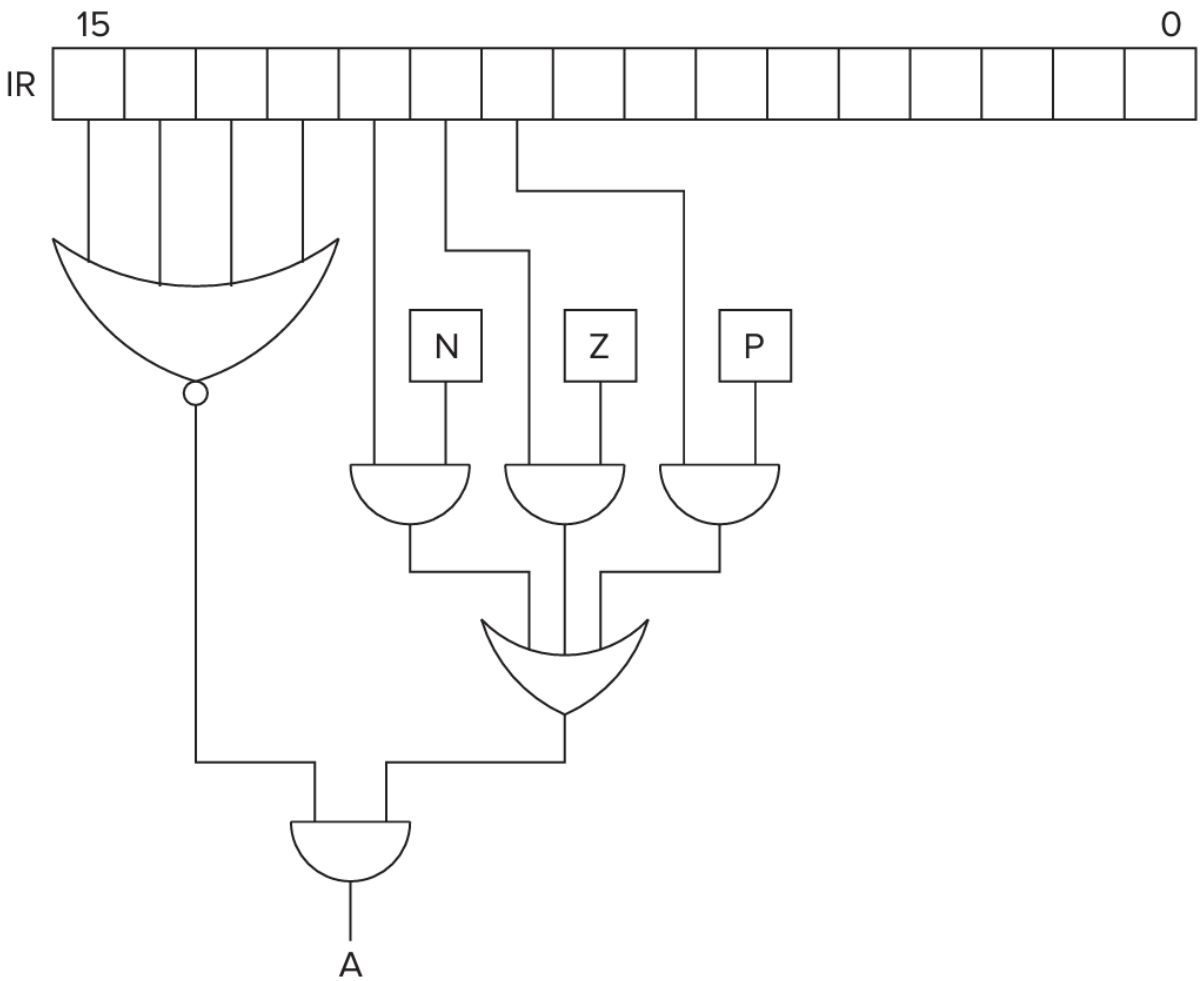
```
NOT R4 R1
NOT R5 R2
AND R6 R4 R5
NOT R3 R6
```

(2): `1001 100 001 111111`

(4): `1001 011 110 111111`

T9

The following logic diagram shows part of the control structure of the LC-3 machine. What is the purpose of the signal labeled A?



> anwser:
>
> 仅当 `IR[15:12]=0000`，即指令为 `BR` 指令，且指令选择的条件码至少有一位为 1 时，信号 A 才为 1，故 A 的作用是判断 `BR` 指令是否进行分支跳转。

T10

The code below is an program to multiply two positive integers in `R1` and `R2` respectively. The result is written into `R0`.

Adress	Data	Operation
x3000	0101 0000 0010 0000	`AND R0 <- R0, #0`
x3001	0001 0010 0111 1111	`ADD R1 <- R1, #-1`
x3002	0000 1000 0000 0010	`BRn x3005`
x3003	0001 0000 1000 0000	`ADD R0 <- R2, R0`
x3004	0000 1111 1111 1100	`BRnp x3001`
x3005	1111 0000 0010 0101	`HALT`

What results will be stored in `R0` if we replace the instruction in x3003 with **`ADD R0 <- R0, R1`**? Use `R1` or `R2` to represent your answer.

answer:

Adress	Data	Operation	Result
x3000	0101 0000 0010 0000	`AND R0 <- R0, #0`	`R0=0`
x3001	0001 0010 0111 1111	`ADD R1 <- R1, #-1`	`R1=R1-1`
x3002	0000 1000 0000 0010	`BRn x3005`	`if R1 < 0, HALT`
x3003	0001 0000 1000 0000	`ADD R0 <- R2, R0`	`R0=R2+R0`
x3004	0000 1111 1111 1100	`BRnp x3001`	`loop`
x3005	1111 0000 0010 0101	`HALT`	

程序累加 `R1` 次 `R2` 的值，实现乘法。其中，`x3001` 处实现 `R1` 递减，`x3003` 处实现 `R2` 的累加。如果把 `x3003` 处的指令换成 `ADD <- R0, R1`，变成了累加“`R1`”，但要注意，`R1` 的值已经被递减过了，所以最后的得到的是从 0 累加到 `R1` 的值，即 $\frac{R1\cdot(R1-1)}{2}$。